EUROTeV-Report-2006-071

AN ILC MAIN LINAC SIMULATION PACKAGE BASED ON MERLIN

D. Kruecker, F. Poirier, N. Walker, DESY, Hamburg, Germany

July 07, 2006

Abstract

The preservation of the ultra-small vertical emittance in the International Linear Collider (ILC) will require the use of beam-based alignment techniques, the expected performance of which relies heavily on the use of simulation tools. In this report, we present the newest release of a purpose-built ILC main linac simulation tool, based on the ***Merlin*** C++ class library [1]. Examples of results from Dispersion Free Steering (DFS) simulations are also be presented.

AN ILC MAIN LINAC SIMULATION PACKAGE BASED ON MERLIN

D. Krücker*, F. Poirier*, N. Walker*, DESY, Hamburg, Germany

Abstract

The preservation of the ultra-small vertical emittance in the International Linear Collider (ILC) will require the use of beam-based alignment techniques, the expected performance of which relies heavily on the use of simulation tools. In this report, we present the newest release of a purpose-built ILC main linac simulation tool, based on the ***Merlin*** C++ class library [1]. Examples of results from Dispersion Free Steering (DFS) simulations are also be presented.

INTRODUCTION

The International Linear Collider (ILC) requires the generation and subsequent preservation of ultra-small vertical emittance beams. The extremely tight alignment tolerances required to avoid emittance dilution due to spurious dispersive effects, mandates the use of Beam Base Alignment (BBA) and other beam-based tuning emittance tuning techniques. Dispersion Free Steering (DFS) [2] is one now relatively well established BBA technique, primarily foreseen as one of the basic tools for use in the superconducting linac.

The emittance tuning has historically relied on the use of simulation tools. Over the last decade these tools have progressively become faster and more sophisticated, allowing ever more realistic modelling of the accelerator. One of the goals of the ILC beam-dynamics community is to provide full so-called 'start-to-end' simulations of the collider, which includes both the initial static tuning (BBA), emittance and ultimately luminosity tuning, in a simulated realistic environment containing both static and dynamic (i.e. time-dependent) errors.

As part of this on-going effort, we present here the first release of a new package for simulating DFS in the main linac of the ILC. The application is written in C++ and has been implemented using the ***Merlin*** Class Library for Accelerator Simulations [1].

THE FUNDAMENTAL BBA ALGORITHM

Dispersion Free Steering – or DFS as it is known – refers to a class of BBA algorithms which attempt to locally correct the anomalous dispersion arising from magnet and other accelerator component alignment errors. The DFS algorithms have been developed by many groups over the years and existing studies are well documented.

Although manifestly similar in approach, the DFS studies made to date by different groups tend to differ in the exact details of how the algorithm is applied. These differences have often been quoted as the reason why results have not quantitatively agreed in the past. With this in mind, ILCDFS has been designed to support many variants of DFS, with a view to making direct comparisons of the different possible approaches.

The basic DFS algorithm attempts to correct an off-energy trajectory to the design value. This is achieved by recording the difference between an on-energy and off-energy trajectories, and finding a steering solution such that the following figure of merit is minimised:

$$\chi^2 = \sum_i w_{diff}^2 \left[\Delta y_i(\delta) - \Delta\tilde{y}_i(\delta)\right]^2 + w_{abs}^2 \left[y_i(0) - \tilde{y}_i(0)\right]^2 ,$$

where $\Delta y_i(\delta) = y_i(\delta) - y_i(0)$ is the measured (simulated) difference trajectory at the i^{th} monitor for an energy deviation δ. The tilda ~ indicates the *design* or *goal* value. The second term is a soft constraint on the absolute (on-energy) trajectory. w_{diff} and w_{abs} are the weights for the difference and absolute trajectories respectively ($w_{diff} >> w_{abs}$). The standard approach is to divide the main linac into several overlapping *segments* (sometimes referred to as *bins*), and correct each segment in turn.

Where the differences in algorithms appear can be broadly divided into two categories:

1. energy adjustment approach;
2. segment boundary corrections.

A further difference is the basic beam dynamics models used for simulating the beam in the main linac. In the ILCDFS framework, these three fundamental concepts are encapsulated as abstract C++ classes, allowing them to be easily replaced and modified.

ILCDFS PHILOSOPHY AND STRUCTURE

Figure 1 shows the high-level class structure of ILCDFS. The central (control) class is DFSApp, of which there is only one instantiation in the application. DFSApp is responsible for coordinating the simulation, and iterating the DFS algorithms over the segments of the accelerator. Class DFSCorrection encapsulates the DFS algorithm itself. An object of class DFSCorrection is associated with a unique accelerator segment, and is responsible for:

- calculating the required model response matrix to be applied for the correction (using SVD);
- recording the necessary simulated data; and
- using the above model, calculating a correction and applying it.

Energy modification

DFSCorrection must know how to adjust the energy of the accelerator in order to (a) construct the correction model, and (b) to record the simulated data. The abstract class EnergyAdjustmentPolicy encapsulates exact details

* Work supported by the Commission of the European Communities under the 6th Framework Programme "Structuring the European Research Area", contract number RIDS-011899.

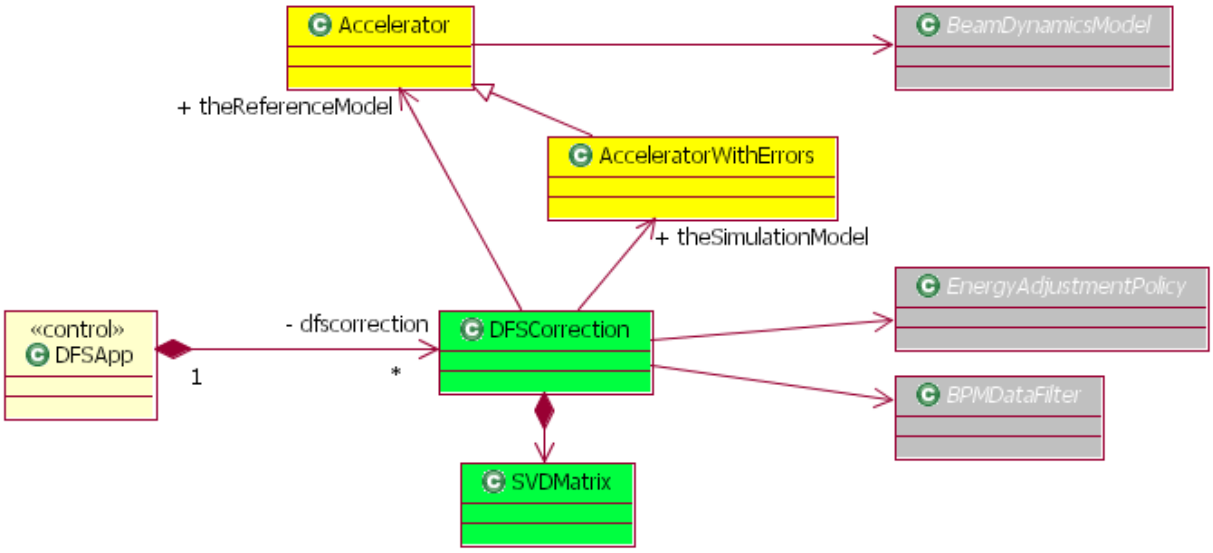

Figure 1 Basic class diagram (UML) showing the top-level structure of ILCDFS. The abstract classes on the right of the diagram encapsulate the specific details of the algorithm being simulated.

of how the energy modification is made. (Note that the exact same method is used for constructing the correction model as is used for acquiring the simulated data.) A concrete EnergyAdjustmentPolicy can support an arbitrary number of energy *states*: state 0 is by default the on-energy machine, while states 1, 2, 3… can represent measurements of trajectories with differing energy

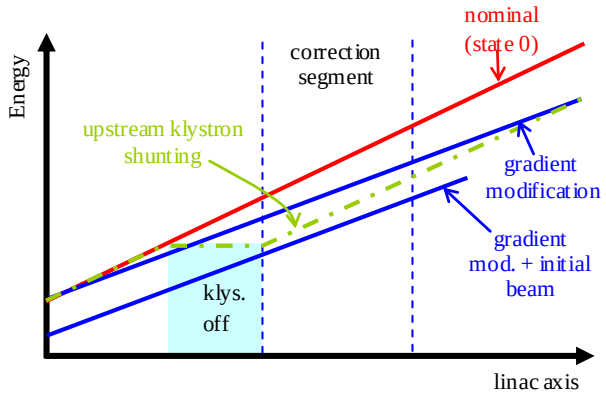

Figure 2 Examples of possible approached to DFS energy adjustment.

configurations. An EnergyAdjustmentPolicy has access to all the klystrons in the accelerator, as well as the beams being tracked (one for each required state). The latter enables a concrete policy to directly (artificially) modify the beam energy or phase. Figure 2 shows examples of possible energy adjustment policies, while Figure 3 shows some example results.

Segment boundary correction

Since DFS is applied piecewise to the accelerator segments. Several published approaches have advocated correcting for the difference in initial phase space coordinates at the segment entrance; this has been considered necessary to mitigate beam jitter effects, anomalous steering from cavities etc. In ILCDFS this concept has been generalised into a data filter which is applied to the recorded trajectories before the DFS correction is calculated (abstract class BPMDataFilter)[*].

ACCELERATOR MODELS AND ERRORS

Class Accelerator encapsulates all the necessary interfaces and details associated with the accelerator model, including applying errors and simulating the tracking of the beam (see section on Beam Dynamics Models below). Class Accelerator provides a *control system type* interface to the DFS algorithms classes, which reflects the physical situation in the control room.

ILCDFS contains two models of the accelerator to be simulated. The *reference* model represents the error-free design accelerator, which is used by DFSCorrection to calculate the correction model and design trajectories. The *simulation* model represents an instantiation of the physical accelerator including errors. The class AcceleratorWithErrors (derived from class Accelerator) contains methods for specifying and applying a wide range of alignment, field and diagnostic errors. Generally, several 'seeds' of random errors are generated for the simulated model.

BEAM DYNAMICS MODELS

Class Accelerator supports several methods for tracking a beam, which are used by DFSCorrection when applying the algorithm (or calculating the correction model). The actual tracking of the beam is delegated to an object of the

[*] As of writing, the beta release of ILCDFS contains no concrete BPMDataFilter implementations.

abstract class BeamDynamicsModel. This mechanism allows the 'plug and play' of different physical models of the beam dynamics to be used. Two concrete BeamDynamicsModel classes are currently supported:

- ParticleTrackingModel: represents a beam as an ensemble of particles which are tracked using ray tracing through the accelerator. A ParticleTrackingModel with a single particle can be used to approximately simulate the beam centroid.
- SMPTrackingModel: represents a beam as a collection of so-called sliced macro-particles, each of which represents the first- and second-order moments of that slice of the bunch. The former is generally more accurate but significantly slower than the latter (which is the preferred approach for linac studies). The BeamDynamicsModel associated with both the reference and simulation models can be changed. DFSApp contains three such models: a beam dynamics model which is used by the:

- reference model for calculation of the correction model (response matrices).
- simulation model during the simulated data acquisition for DFS correction
- simulation model during the final estimation of the emittance after the correction is complete.

The current example in the beta-release uses a single-particle ParticleTrackingModel for the first two, and an SMPTrackingModel for the last. The use of a single ray-tracing greatly speeds up the calculation of the response matrices for the correction model, as well as the application of the correction itself. SMPTrackingModel is used to give an accurate estimate of the final emittance (including wakefield effects).

IMPLEMENTATION

The classes discussed above represent an application level design specific to the ILCDFS. The class structure was arrived at after careful analysis of the problem at hand. Each of these (and the remaining) classes in the current ILCDFS have been implemented using classes from the ***Merlin*** C++ class library. In most cases, the complexity of the ***Merlin*** classes has been hidden by wrapping them with the ILCDFS application classes, which generally contain cleaner use-driven interfaces. Some examples of implementation details are:

- Accelerator contains a pointer to the ***Merlin*** class *AcceleratorModel*, which supplies the underlying structure and functionality.
- The concrete BeamDynamicsModel classes use the ***Merlin*** beam dynamics implementations *ParticleTracker* and *SMPTracker* (together with their associated *Bunch* types).
- DFSCorrection makes use of ***Merlin*** *ROChannel* and *RWChannel* classes for generic access to correctors and BPMs.
- EnergyAdjustmentPolicy uses the ***Merlin*** *Klystron* and *ReferenceParticle* classes.

EXAMPLE RESULTS

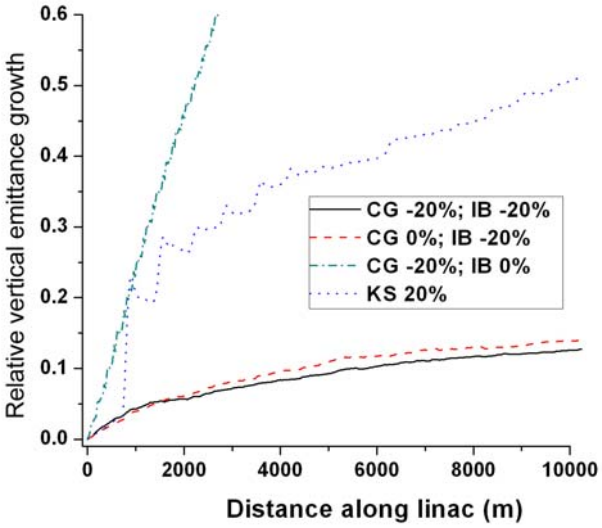

Figure 3 Simulated vertical emittance growth along the linac for various energy adjustment methods: CG = constant gradient; IB = initial beam energy; KS = klystron shunting (see figure 2). The results are averaged over 100 seeds of random misalignments [3]

AVAILABILITY

The current ILCDFS beta-release – together with the ***Merlin*** class library are available from the ***Merlin*** web site [1].

REFERENCES

[1] ***Merlin*** *– A C++ Class Library for Accelerator Simulations;* http://www.desy.de/~merlin.

[2] T. Raubenheimer, R. Ruth, Nucl. Instrum. Meth. A302:191-208,1991

[3] F. Poirier *et al*, EUROTeV report (in preparation); see also D. Schulte *et al*, these proceedings (MOPLS098).